숭실대학교
Soongsil University

프로그램 설명서

과 목	컴퓨터 비전
제 출 일 자	2020-05-22
담 당 교 수	김계영
학 과	소프트웨어학부
학 번	20150283
이 름	최성률

목차

1. 개요 ---------- 3

1.1 영상 처리

2. 설계 ---------- 4

2.1 학습 & 인식 대상 선정

3. 구현 ---------- 5

3.1 함수 기능

4. 결과 ---------- 15

4.1 실행 과정

4.2 결과

1. 개요

1.1 영상 처리

1. 영상을 회색조(Grayscale)로 처리한다.
2. 회색조(Grayscale)로 처리된 영상을 이진화(Binarization)로 분할한다. 분할 처리 과정에서는 임계값은 Otsu thresholding algorithm으로 코드를 구현하여 배경과 객체를 구별되게 한다.
3. 이진화로 분할 처리된 영상을 잡음(Noise)을 처리하기 위해 형태학적 연산 기법(Morphological operation)으로 필터링(Filtering)을 진행한다. 처리 과정은 침식(Erosion)한 후 팽창(Dilation)하는 Opening기법(or Closing기법)을 사용하여 잡음을 제거한다. 과정에서 Kernel size는 다음과 같다.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Kernel size(5 X 5)

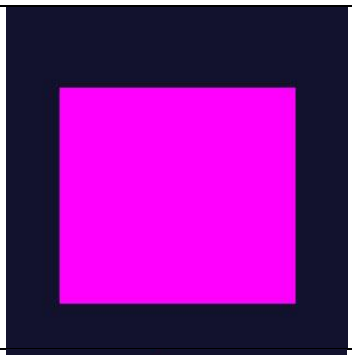
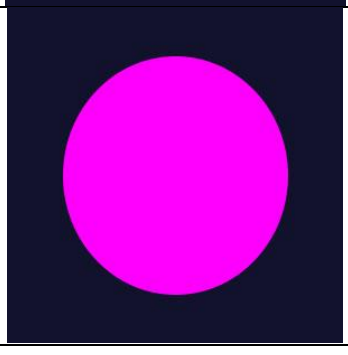
4. 잡음 처리가 된 영상을 레이블링(Labeling)을 진행한다. 레이블링은 컨투어 기반 알고리즘(Contour based algorithm)으로 처리한다. 이 과정에서 경계(Boundary)를 추출한다.
5. 경계(Boundary)의 (x, y)좌표를 해당 좌표와 정해진 Window size(Contour size / 8)로 일정 범위에 떨어진 두 좌표로 이은 직선과의 유클리디언 거리(Euclidean distance)를 구하여 특징을 추출한다.
6. 특징을 추출하고 클래스 여러 영상과 SDTW(Statistical Dynamic Time Warping)을 진행하여 평균 패턴과 표준 편차를 구한다.
7. 인식 영상도 영상 처리 과정(1~5)을 진행하고 클래스 별로 평균 패턴과의 DTW(Dynamic Time Warping)을 진행하여 비유사도(Dissimilarity)를 구하여 인식 영상을 비유사도가 가장 작은 클래스로 인식한다.

2. 설계

2.1 학습 & 인식 대상 선정

특징을 Localized Contour Sequence로 추출하여 구별도가 크도록 꼭지점의 개수가 다른 기본 도형을 선정하였다.

1. 학습을 같은 색깔의 이미지로 진행하고, 인식은 다른 색깔의 도형을 인식할 수 있도록 한다.
2. 색깔을 섞어서도 학습을 진행하고, 인식을 진행한다.

	**1. 사각형 도형(Square)** ➔ 꼭지점의 개수: 4개
	**2. 꽃 모양 도형(Flower)** ➔ 꼭지점의 개수: 12개
	**3. 타원 도형(Ellipse)** ➔ 꼭지점의 개수: ∞개

3. 구현

3.1 함수 기능

```
for (int j = 0; j < gRows; j++) {
        for (int i = 0; i < gCols; i++) {
                b = tmpImg.at<Vec3b>(j, i)[0];
                g = tmpImg.at<Vec3b>(j, i)[1];
                r = tmpImg.at<Vec3b>(j, i)[2];

                gray = r * 0.2126 + g * 0.7152 + b * 0.0722;
                grayScale.at<uchar>(j, i) = (uchar)gray;
        }
}
```

<회색조(Grayscale) 처리>

- 각 R, G, B 채널에 특정 가중치 값(R : 0.2126, G : 0.7152, B : 0.0722)을 곱하여 Grayscale 영상으로 처리하였다.

```
-   // Histogram
-   for (int j = 0; j < oRows; j++) {
-    for (int i = 0; i < oCols; i++) {
-            idt[(int)otsu.at<uchar>(j, i)]++;
-    }
-   }
-   // N-Histogram + Total-Mean
-   for (int i = 0; i < 256; i++) {
-    pIdt[i] = idt[i] / ((double)oRows*oCols);
-    tMean += pIdt[i] * i;
-   }
-   // otsu's thresholding :: i = threshold
-   for (int i = 0; i < 256; i++) {
-    // weight initialize
-    wgt_0 = 0.0;
-    mean_0 = 0.0;
-    for (int j = 0; j <= i; j++) {
-            wgt_0 += pIdt[j];
-            mean_0 += pIdt[j] * j;
-    }
-   wgt_1 = 1.0 - wgt_0;
-    mean_1 = tMean - mean_0;
```

```
    -   // mean initialize
    -   if (wgt_0 != 0) {
    -           mean_0 /= wgt_0;
    -   }
    -   if (wgt_1 != 0) {
    -           mean_1 /= wgt_1;
    -   }
    -           double next_result = (wgt_0 * (mean_0 - tMean) * (mean_0 - tMean)) + (wgt_1
        (mean_1 - tMean) * (mean_1 - tMean));
    -   if (result <= next_result) { // continue
    -           result = next_result;
    -   }
    -   else { // find threshold :: i-1(maximum result) + break
    -           for (int b = 0; b < oRows; b++) {
    -                   for (int a = 0; a < oCols; a++) {
    -                           if (otsu.at<uchar>(b, a) < i) { // i-1
    -                                   otsu.at<uchar>(b, a) = 0;
    -                           }
    -                           else {
    -                                   otsu.at<uchar>(b, a) = 255;
    -                           }
    -                   }
    -           }
    -   break; // exit first for-loop
    -   }
}
```

<이진화(Binarization) 처리 함수>

- Otsu threshold algorithm을 사용하여 임계값을 정하고 흑백(0 or 255)값으로 영상을 처리하였다.

```
    -   for (int j = 0; j < dRows; j++) {
    -           for (int i = 0; i < dCols; i++) {
    -                   for (int b = 0; b < kRows; b++) {
    -                           for (int a = 0; a < kCols; a++) {
    -                                   tmpJindex = j + (b - kMidIndex);
    -                                   tmpIindex = i + (a - kMidIndex);
    -                                   if (tmpJindex >= 0 && tmpJindex < dRows &&
        tmpIindex >= 0 && tmpIindex < dCols) {
    -                                           if (kernel.at<uchar>(b, a) == 255) {
    -                                                   check |= img.at<uchar>(tmpJindex,
        tmpIindex);
```

```
-                                               }
-                                           }
-                                       }
-                                   }
-                                   dilImg.at<uchar>(j, i) = check;   // dilation
-                                   check = 0;
-                           }
-                   }
```

<팽창(Dilation) 처리 함수>

- Kernel을 매개 변수로 받아 Kernel의 중앙 인덱스를 필터링할 좌표 인덱스와 대칭시켜서 팽창을 처리한다.

```
-   for (int j = 0; j < eRows; j++) {
-           for (int i = 0; i < eCols; i++) {
-                   for (int b = 0; b < kRows; b++) {
-                           for (int a = 0; a < kCols; a++) {
-                                   tmpJindex = j + (b - kMidIndex);
-                                   tmpIindex = i + (a - kMidIndex);
-                                   if (tmpJindex >= 0 && tmpJindex < eRows &&
    tmpIindex >= 0 && tmpIindex < eCols) {
-                                           if (kernel.at<uchar>(b, a) == 255) {
-                                                   check &= img.at<uchar>(tmpJindex,
    tmpIindex);
-                                           }
-                                   }
-                           }
-                   }
-                   eroImg.at<uchar>(j, i) = check;   // erosion
-                   check = 255;
-           }
-   }
```

<침식(Erosion) 처리 함수>

- Kernel을 매개 변수로 받아 Kernel의 중앙 인덱스를 필터링할 좌표 인덱스와 대칭시켜서 침식을 처리한다.

```
-   openImg = Img2erosion(img, kernel);
-   openImg = Img2dilation(openImg, kernel);
```

<Opening 처리 함수>

- 침식 -> 팽창으로 Opening 처리한다.

```
-   closeImg = Img2dilation(img, kernel);
-   closeImg = Img2erosion(closeImg, kernel);
```

<Closing 처리 함수>

- 팽창 -> 침식으로 Closing 처리한다.

```
-   for (int j = 0; j < cRows; j++) {
-           originImg.at<uchar>(j, 0) = 0;
-           originImg.at<uchar>(j, cCols - 1) = 0;
-   }
-
-   for (int i = 0; i < cCols; i++) {
-           originImg.at<uchar>(0, i) = 0;
-           originImg.at<uchar>(cRows - 1, i) = 0;
-   }
-
-   // Display
-   Mat contourImg = Mat::zeros(cRows, cCols, CV_8UC1);
    // Contour Display 하기 위한 이미지
-   labelImg = LabelingwithBT(originImg, contourImg);
    // labelImg : 라벨링 결과 이미지
```

```
for (int j = 1; j < (bRows - 1); j++) {
                for (int i = 1; i < (bCols - 1); i++) {
                        cur_p = binaryImg.at<uchar>(j, i);
                        if (cur_p == 255 && labelImg.at<uchar>(j, i) == 0) {       //
object
                                ref_p1 = labelImg.at<uchar>(j, i - 1);
                                ref_p2 = labelImg.at<uchar>(j - 1, i - 1);
                                if (ref_p1 > 1) {        // propagation
                                        labelImg.at<uchar>(j, i) = ref_p1;
                                }
                                else if (ref_p1 == 0 && ref_p2 >= 2) {    // hole
                                        labelImg.at<uchar>(j, i) = ref_p2;
                                        BTracing8(j, i, ref_p2, BACKWARD, binaryImg,
labelImg, contourImg,  LUT_BLabeling);
                                }
                                else if (ref_p1 == 0 && ref_p2 == 0) {    // region start
                                        labelImg.at<uchar>(j, i) = ++labelnumber;
                                        BTracing8(j, i, labelnumber, FOREWARD, binaryImg,
```

```
labelImg, contourImg, LUT_BLabeling);
                                }
                        }
                }
        }
```

<Labeling 처리 함수>

- 사전 처리로 바깥쪽은 모두 0으로 초기화하고 전파, 구멍, 시작점을 이용하여 레이블링을 처리한다.

```
-   do {
-           // Display(Gray Color)
-           boundImg.at<uchar>(y, x) = 127;
-           // for LCSs
-           _xy.first = y;
-           _xy.second = x;
-           _xyList.push_back(_xy);
-
-           Read_neighbor8(y, x, neighbor8, binImg);
-           start_o = (8 + cur_orient - 2) % 8;
-           for (i = 0; i < 8; i++) {
-                   add_o = (start_o + i) % 8;
-                   if (neighbor8[add_o] == 255) { break; }
-           }
-           if (i < 8) {
-                   CalcOrd(add_o, &y, &x);
-                   cur_orient = add_o;
-           }
-           if (LUT_BLabeling[pre_orient][cur_orient] == 1) {
-                   labImg.at<uchar>(_y, _x) = label;
-           }
-           _y = y;
-           _x = x;
-           pre_orient = cur_orient;
-   } while ((y != end_y) || (x != end_x));
```

<Contour tracing 처리 함수>

- 시작 좌표 기준으로 구멍은 9시부터 시계방향으로 돌고, 나머지는 3시부터 시계방향으로 돌며 경계를 추적한다.

```
-   // h(i) 구하기
```

```
-   for (int i = 0; i < contourList.size(); i++) {
-           if (contourList.at(i).size() > 8) { tempWindow = contourList.at(i).size() /
    8; }
-           else{tempWindow = 1;}
-           window = (tempWindow % 2 == 0) ? tempWindow + 1 : tempWindow;
            // window size =  Contour size / 8의 홀수
-           for (int j = 0; j < contourList.at(i).size(); j++) {
-                   LCS[lcsIndex++] = FindDistance(i, j, window);
-           }
-
-   }
```

```
dist = fabs(_y * (_xMinus - _xPlus) + _x * (_yPlus - _yMinus) + (_xPlus * _yMinus) - (_yPlus
* _xMinus))
                / sqrt((_xMinus - _xPlus) * (_xMinus - _xPlus) + (_yPlus - _yMinus) *
(_yPlus - _yMinus));
```

<LCS(Localized Contour Sequence) 추출 함수>

- Window size는 경계 크기의 1/8이며 해당 좌표와 Window size의 범위의 직선과의 수직 거리를 식으로 정리하여 LCS를 추출하였다.

```
-   // Initialization
-   dMat[0][0] = fabs(LCS1[0] - LCS2[0]);
-   if (deviPatt && deviPatt[0]) { dMat[0][0] /= deviPatt[0]; }
-   gMat[0][0] = 0;
-
-   for (int j = 1; j < size2; j++) {
    // j축(가로 축) : 열
-           dMat[0][j] = fabs(LCS1[0] - LCS2[j]);
-           if (deviPatt && deviPatt[0]) { dMat[0][j] /= deviPatt[0];}
-           dMat[0][j] += dMat[0][j - 1];
-           gMat[0][j] = 2;
-   }
-   for (int i = 1; i < size1; i++) {
    // i축(세로 축) : 행
-           dMat[i][0] = inf;
-   }
-
-   // Forward
-   for (int i = 1; i < size1; i++) {
-           for (int j = 1; j < size2; j++) {
-                   dMat[i][j] = fabs(LCS1[i] -  LCS2[j]);
-                   if (deviPatt && deviPatt[i]) { dMat[i][j] /= deviPatt[i]; }
-                   gMat[i][j] = ForDTW_MinArg(i, j, dMat);
-           }
```

```
-    }
-
-    // Backtracking
-    int i = size1 - 1;
-    int j = size2 - 1;
-    int k = 0;
-
-    while (i != 0 && j != 0) {
-            _path.push_back(gMat[i][j]);
-            switch (gMat[i][j]) {
-                    case 1:  i--; k++; break;
-                    case 2: j--; k++; break;
-                    case 3:  i--; j--; k++; break;
-            }
-    }
-
-    // Termination
-    disS = dMat[size1 - 1][size2 - 1] / ((double)k + 1);
```

<DTW(Dynamic Time Warping) 처리 함수>

- 일반적으로 표준 편차가 없을 경우, 두 Localized Contour Sequence간의 절댓값 차를 이용하여(Manhattan distance) 비유사도를 구하고 매칭 좌표들을 기록하기 위해 Path를 기록한다.
- 인식 과정에서의 Dynamic Time Warping은 두 패턴의 절대값 차를 표준 편차로 나누어 주어 마할라노비스 거리(Mahalanovis distance)로 비유사도를 측정한다.

```
-   // Initialize : 처음 평균 패턴은 가장 작은 LCS Size의 이미지
-   for (int i = 0; i < pattCnt; i++) {
-           newPatt[i] = curPatt[i] = trainImgLCSs[standardIndex][i];
-   }
-
-   // Statistical Dynamic Time Warping
-   while (isChange) {
-           for (int i = 0; i < imgCnt; i++) {
-                   DTWarping(curPatt, trainImgLCSs[i], nullPtr);
-           }
-           ForSDTW_Average(trainImgLCSs, curPatt);
    // 새로운 평균 패턴 생성 + 매칭 위치별로 분산
-
```

```
-           if (FindThres(curPatt, newPatt) < 1) { isChange = false; } // 평균 패턴이
    변하는지 검사
-           else { _imgPath.clear(); }
-
-           for (int a = 0; a < pattCnt; a++) {
-                   newPatt[a] = curPatt[a];
                           // 새로운 평균 패턴
-           }
-   }
-   ForSDTW_Deviation(trainImgLCSs, newPatt, deviPatt);
                   // 분산
```

```
for (int i = 0, j = 0; i < imgCnt; i++) {
                avgPatt[0] += trainImgLCSs[i][0];
                matchCnt[0]++;
                mIndex = 0;
                for (int k = _imgPath.at(i).size() - 1; k >= 0; k--) {
        // 이미지당 path size만큼
                        switch (_imgPath.at(i).at(k)) {
                        case 1:
                                j++;
                                matchCnt[j]++;
                                avgPatt[j] += trainImgLCSs[i][mIndex];
                                break;
                        case 2:
                                mIndex++;
                                matchCnt[j]++;
                                avgPatt[j] += trainImgLCSs[i][mIndex];
                                break;
                        case 3:
                                j++; mIndex++;
                                matchCnt[j]++;
                                avgPatt[j] += trainImgLCSs[i][mIndex];
                                break;
                        }
                }
                j = 0;
        }

        for (int j = 0; j < pattCnt; j++) {
                avgPatt[j] /= ((double)matchCnt[j] + 1);
        }
```

```
for (int i = 0, j = 0; i < imgCnt; i++) {
                deviPatt[0] += (avgPatt[0] - trainImgLCSs[i][0]) * (avgPatt[0] -
trainImgLCSs[i][0]);
                matchCnt[0]++;
                mIndex = 0;
                for (int k = _imgPath.at(i).size() - 1; k >= 0; k--) {
        // 이미지당 path size만큼
                        switch (_imgPath.at(i).at(k)) {
                        case 1:
```

```
                                j++;
                                matchCnt[j]++;
                                deviPatt[j] += (avgPatt[j] - trainImgLCSs[i][mIndex]) *
(avgPatt[j] - trainImgLCSs[i][mIndex]);
                                break;
                        case 2:
                                mIndex++;
                                matchCnt[j]++;
                                deviPatt[j] += (avgPatt[j] - trainImgLCSs[i][mIndex]) *
(avgPatt[j] - trainImgLCSs[i][mIndex]);
                                break;
                        case 3:
                                j++; mIndex++;
                                matchCnt[j]++;
                                deviPatt[j] += (avgPatt[j] - trainImgLCSs[i][mIndex]) *
(avgPatt[j] - trainImgLCSs[i][mIndex]);
                                break;
                        }
                }
                j = 0;
        }

        for (int j = 0; j < pattCnt; j++) {
                deviPatt[j] = sqrt(deviPatt[j] / (double)matchCnt[j]);
        }
```

```
for (int i = 0; i < pattCnt; i++) {
                thres += fabs(curPatt[i] - prePatt[i]);
        }
        thresArray.insert(thresArray.begin(), thres);

        printf("-----------------Threshold = %lf--------------₩n", thres);

        if (thresArray.size() > 100) {
                sortThres.assign(thresArray.begin(), thresArray.end());
                sort(sortThres.begin(), sortThres.end());
                if (sortThres.back() / sortThres.front() < 2 || sortThres.at(0) ==
sortThres.at(1) || sortThres.at(8) == sortThres.at(9)) { return 0.0; }
                sortThres.clear();
                thresArray.pop_back();
        }
```

<SDTW(Statistical Dynamic Time Warping) 처리 함수>

- 가장 작은 Localized Contour Sequence size를 가지는 영상 패턴을 기준으로 하여 각 클래스 안의 패턴들과 Dynamic Time Warping을 진행한다. 새로운 패턴은 매칭 위치별로의 평균 값으로 패턴을 만든다.

- 반복문을 탈출하기 위한 조건은 임계값 설정이다.

  ①이전 평균 패턴과 새로운 평균 패턴과의 절댓값 차가 1이하이거나

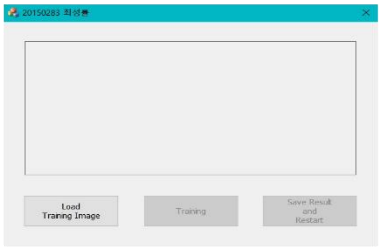
②100번 이상의 평균 패턴을 생성하였으면 임계값을 저장한 리스트를 오름차순으로 정렬하여 같은 값이 존재하거나 현재 임계값과 100번 전의 임계값과의 차가 1이하일 때 탈출한다.

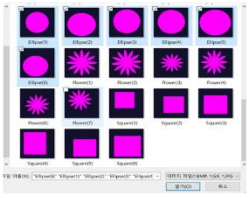
4. 결과

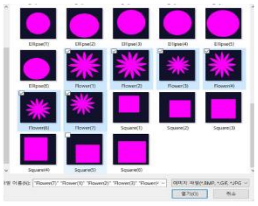
4.1 실행 과정

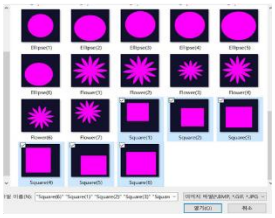
<프로그램 실행 및 학습 이미지 불러오기>

주의! 클래스별로 불러오기(선택은 최대 10장까지만 가능)

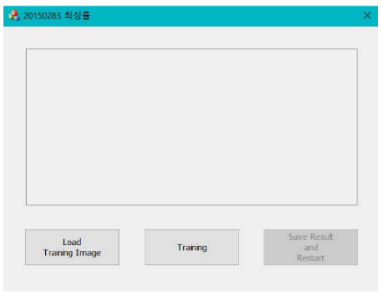


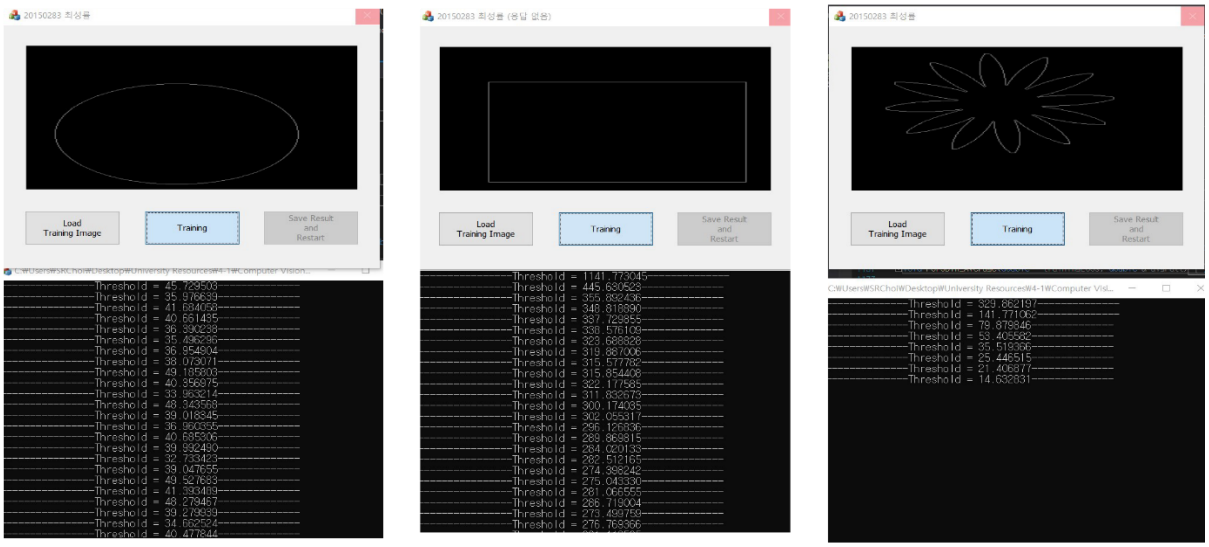






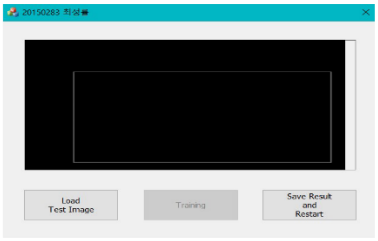
<학습 시작(버튼 클릭)>

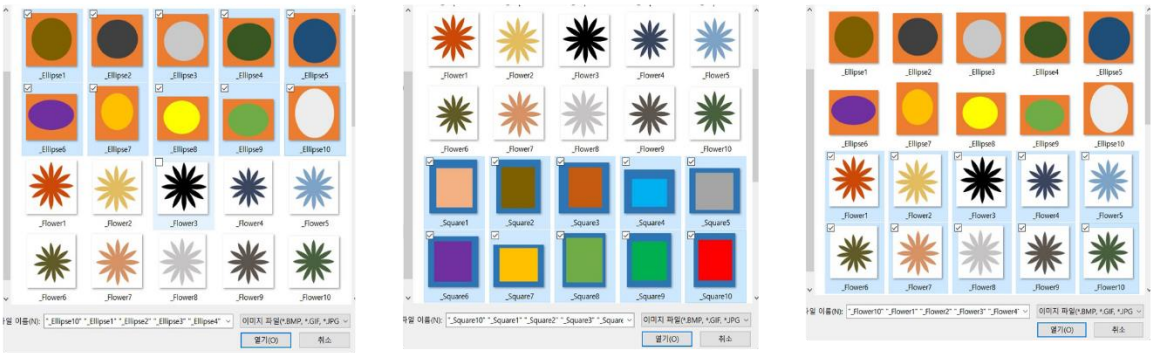




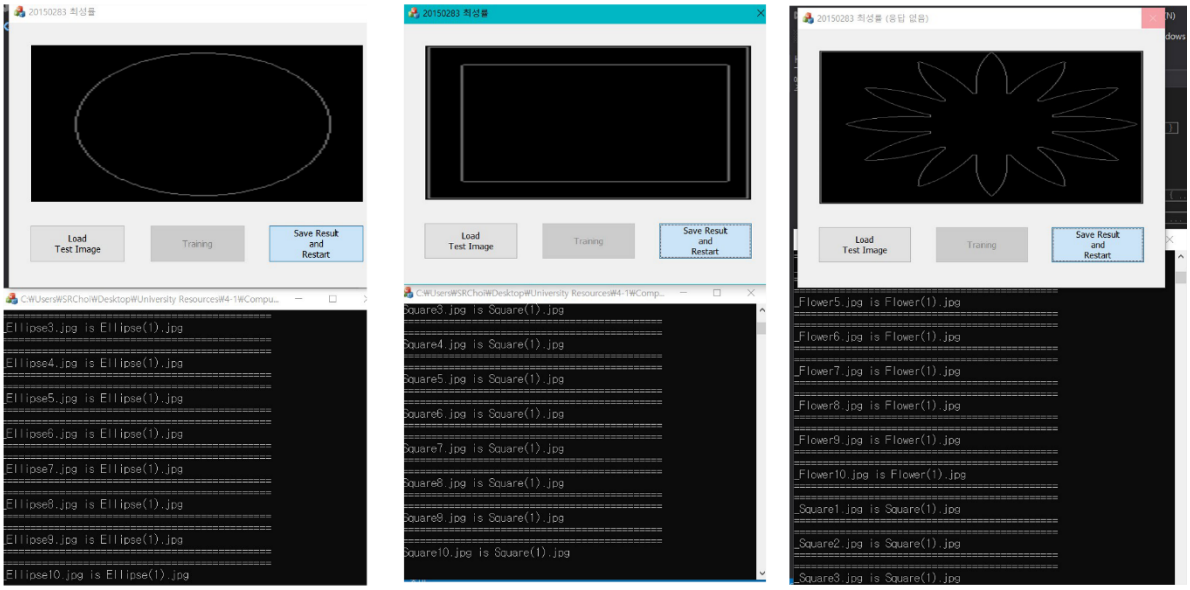
<인식 이미지 불러오기>

주의! 선택은 최대 10장까지만 가능(대화상자당)





<인식 시작(버튼 클릭)>



4.2 결과

Reference Images (6 each)	Test Images (10 each)		
	Class 1(Ellipse)	Class 2(Square)	Class 3(Flower)
Class 1(Ellipse)	**10**	0	0
Class 2(Square)	0	**10**	0
Class 3(Flower)	0	0	**10**

<학습 이미지 : 10장씩, 인식 이미지 6장씩>

➔ **인식률: 100%**

Reference Images (10 each)	Test Images (6 each)		
	Class 1(Ellipse)	Class 2(Square)	Class 3(Flower)
Class 1(Ellipse)	**6**	0	0
Class 2(Square)	0	**6**	0
Class 3(Flower)	0	0	**6**

<학습 이미지 : 10장씩, 인식 이미지 6장씩>

➔ **인식률: 100%**

결과: 색깔을 섞어서도 학습을 진행하여도 인식률이 100%의 결과를 초래하였다. Localized Contour Sequence 길이는 꼭지점의 개수가 차이가 있을 때, 높은 인식률을 보임을 알 수 있다.